Apple Assembly Line

$1.80

Volume 7 -- Issue 8 May, 1987

The Chips are Falling

We have just learned that Western Design Center is dramatically lowering the price of the 65802 microprocessors. We've been selling the '802 for $50.00: Effective about the time you read this our price will be $25.00. All the chips they produce are now rated to operate to at least 4 MHz, and they're now getting a significant yield of 6 and even 8 MHz parts. Evidently the volume production made possible by Apple's use of the 65816 in the IIgs is paying off.

Missing ProDOS Books

We had a scare a couple of weeks ago: Quality Software and Simon & Schuster had both run out of copies of Beneath Apple ProDOS, that excellent reference on the inner workings of ProDOS, so it looked for a while like we might lose a valuable resource. All is well, though, the folks at Quality are planning a new printing, so we expect to have more copies of the book in a month or two. We'll just hold any orders until that time.

Curiously, both Addison-Wesley's ProDOS Technical Reference Manual and Simon & Schuster's Apple ProDOS: Advanced Features for Programmers have been out of stock at the publishers for a couple of months now. A-W tells us that a revised edition of Apple's manual will be published in late June. S & S has Advanced Features on backorder, but won't quote even a tentative delivery date.

A New Pattern Search Monitor Command.....Bob Sander-Cederlof

As I mentioned in my description of the new Apple IIgs monitor commands (AAL Jan 87, pp.23-27), there is a new "P" command which searches through a range of memory for all occurrences of a pattern. Well, I tried to use that command the other day.

First, I couldn't remember with confidence what the syntax was. Then I thought that it must be \pattern\adrl.adr2P. I tried that and it did not work. I looked it up in Fischer's "IIgs Technical Reference", and verified that I did have the correct syntax. It still did not work, and I did need a search capability, so I wrote my own. More on it later. That was last week. I don't know why I didn't look up the command in another reference, but today I did. Both my memory and Fischer's book were wrong!

For some esoteric reason, Apple requires that the "<" character appear between the pattern and the address range. My January article and Apple's own reference material clearly show the "<" there: \pattern\<adrl.adr2P is the correct syntax. Type it correctly, and it works as advertised. For example:

```
     \8E C0\<0/2000.5FFFP will search $2000 through $5FFF
                          in bank 0 for the two bytes $8E
                          and $C0 in successive locations.
     \"Apple"\<FF/0.FFFFP will search all of bank $FF
                          for the string "Apple". (In my
                          IIgs it found that string five
                          times.)
```

If you leave out the "<" character, nothing happens.

The "P" command is a powerful tool, and a welcome addition the Apple monitor. However, I still need some capabilities it does not have. For example, what if I want to find all references to a particular address made using any of the LDA instructions. I don't want to see any references using LDX, LDY, CMP, STA, etc. If I could tell it to only compare certain bits in some bytes, I could accomplish this and many other interesting feats. We might also like the feature of allowing a variable length wild card, such as in FID and FILER filenames and in the S-C Macro Assembler REPLACE command. Maybe I want to include the ability to find PC-relative references to an address. There is probably an endless list of possible features.

I did write a new search program, and did include one of the above mentioned new features. I linked it in through the control-Y vector, so that I can use it from within the monitor. It in effect adds a new command to the monitor, with this syntax "adrl.adr2Ypattern". (Where I show "Y", you have to type control-Y. If you are using the command from within the S-C Macro Assembler, you have to type control-O followed by control-Y.) The pattern is a sequence of hex bytes and/or byte-pairs. A hex byte will have to be completely matched. A byte-pair specifies which bits to compare by giving both a mask byte and a key byte. For example:

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners....................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Source Code of S-C Macro 2.0 (DOS or ProDOS)........each, additional $100
S-C DisAssembler (ProDOS only)...without source code $30, with source $50
RAK-Ware DISASM (DOS only).......without source code $30, with source $50
ProVIEW (ProDOS only)...$20
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
S-C Word Processor (with complete source code)........................$50
DP18 and DPFP, including complete source and object code..............$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)........................($49.95) $45 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $50 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)
Beagle Bros. Applesoft Compiler (ProDOS only)...............($74.95) $65 *

AAL Quarterly Disks.........................each $15, or any four for $45
Each disk contains the source code from three issues of AAL:
Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 25 for $15 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100 *
Envelopes for Diskette Mailers............................ 6 cents each

65802 Microprocessor (Western Design Center)..................($95) $25 ~~$50~~ *
quikLoader EPROM System (SCRG).............................($179) $170 *
PROmGRAMER (SCRG)..($149.50) $140 *
Minuteman 250 Uninterruptible Power Supply...................($359) $320 +

"Apple //gs Technical Reference", Fischer..................($19.95) $19 *
"65816/65802 Assembly Language Programming", Fischer........($21.95) $20 *
"Programming the 65816", Eyes..............................($22.95) $21 *
"Apple //e Reference Manual", Apple Computer................($24.95) $23 *
"Apple //c Reference Manual", Apple Computer................($24.95) $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95) $27 *
"Apple ProDOS: Advanced Features for Programmers", Little..($17.95) $17 *
"ProDOS Inside and Out", Doms & Weishaar...................($16.95) $16 *
"DOStalk Scrapbook", Weishaar & Kersey.....................($14.95) $14 *
"Beneath Apple ProDOS", Worth & Lechner....................($19.95) $18 *
"Beneath Apple DOS", Worth & Lechner.......................($19.95) $18 *
"Inside the Apple //c", Little.............................($19.95) $18 *
"Inside the Apple //e", Little.............................($19.95) $18 *
"Understanding the Apple //e", Sather......................($24.95) $23 *
"Understanding the Apple II", Sather.......................($22.95) $21 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18 *
"Apple][Circuit Description", Gayler.....................($22.95) $21 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95) $18 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95) $18 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95) $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95) $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95) $20 *
"Microcomputer Graphics", Myers............................($14.95) $14 *
"Assembly Lines -- the Book", Wagner.......................($19.95) $18 *
"AppleVisions", Bishop & Grossberger.......................($39.95) $36 *

* On these items add $2.00 for the first item and
$.75 for each additional item for US shipping.
+ Inquire for shipping cost.
Foreign customers inquire for postage needed.
Texas residents please add 7.25 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** Master Card, VISA, Discover and American Express ***

```
0/2000.5FFFYAD F0:80 C0 will find all occurrences
                        of LDA $C080 through $C08F in
                        bank 0, 2000-5FFF.

FF/0.FFFFYC1 DF:D0 DF:D0 DF:CC DF:C5 will find all
                        occurrences of "Apple" without
                        regard to upper or lower case.
```

In the January issue of AAL I also discussed the way to hook into the control-Y vector in the IIgs. It is different from the older Apples. Lines 1160-1230 in the following program install the control-Y vector in a IIgs. In an older Apple we would store only a 2-byte address at $3F9 and $3FA. This will still work in a IIgs, but then you would be limited to running in bank 0 in emulation mode. The bank $E1 vector allows you to be called in native mode, and your code can be in any bank of memory.

When you enter a control-Y command (assuming the vector has been installed) control will come to the SEARCH program. Lines 1260-1290 are a nice general way to save the current mode and status and get into emulation mode. Lines 1600-1630 restore the mode and status before returning.

Lines 1310-1320 call the GET.PATTERN subroutine and quit if that subroutine reports an error. GET.PATTERN scans the input buffer from just after the control-Y to a carriage return. It seemed reasonable to me to require that the entire command line be in the first 127 characters of the input buffer; that left the other half ($280-2FF) free for me to use for storing the converted key and mask bytes. GET.PATTERN stores the key bytes starting at $280, and the mask bytes starting at $2C0. It returns the number of bytes in the key in the X-register. Line 1330 stores this count at WBUF, which happens to also be the beginning of the input buffer.

Lines 1340-1350 copy the bank byte of the address range from the place the monitor keeps it ($E1013E) to the end of a 3-byte pointer in page zero. This seems like a good time to list all the places the monitor puts the addresses it parses. The general form of a monitor command is "a/b<c/d.eX". "a/b" is a target address, with "a" specifying the bank and "b" specifying the address within the bank. "c/d" is the beginning address of a range, with "c" specifying the bank and "d" the address within the bank. "e" is the ending address of the range, in the same bank with "d". "X" represents any monitor command letter. Some commands do not use all of the parsable parts, but you can usually get away with entering them anyway. The monitor subroutine called GETNUM is responsible for parsing the addresses. In older machines GETNUM was at $FFA7. In the IIgs there is still a copy of GETNUM at $FFA7, but it is not used. Instead a new version which starts at $F888 is used. This new version allows the "a/" and "c/" parts. From my analysis, here is where the new GETNUM stores a, b, c, d, and e:

```
a -- $E1/013F
b -- $00/0042,43
c -- $E1/013E
```

d -- $00/003C,3D and $00/0040,41
e -- $00/003E,3F

GETNUM will also leave the pointer into the input buffer which points to the next character after the command character in YSAVE, which is $00/0034.

The IIgs also has a new subroutine at $FCCA, which gets the next character from the input buffer and converts it from lower-case to upper-case if it is a lower-case letter.

Lines 1370-1580 perform the actual search operation. Lines 1370-1440 compare the masked portions of the key bytes with the masked portion of the memory bytes starting at PNTR. If they all match, lines 1460-1490 call on PRADR to print the 3-byte address. Lines 1500-1580 increment the PNTR and test to see if we have come to the end of the range. [Note that a multiple-byte key will be tested beyond the end of the range. That is, the search will continue until the first key byte has been tested throughout the range. This means the key bytes after the first one will extend beyond the end of the range. I am pointing this out so that you will not accidentally start tripping softswitches in the I/O space when you have a range ending at $BFFF.]

There are ample possibilities here for you to expand the features of this search. You could add to the GET.PATTERN a more flexible wild-card scheme. You could allow ASCII strings to be entered. You could automatically protect the I/O softswitch range, especially $C000-C0FF. You could save the current shadowing status and then turn off all shadowing during the search. And so on.

I slipped in another little routine at lines 2150-2420. I am not calling it here, but when I did it printed out the currently parsed addresses. It helped me to get a better feeling for what happens when you enter an incomplete address specification. If you leave off the bank, it uses the same one you used before. If you leave out the range beginning, it begins where the previous command ended. There are a lot of choices here, so I find I usually just type the complete command to be sure.

```
                  1000 *SAVE S.SEARCH.IIGS
                  1010 *--------------------------------
34-               1020 YSAVE  .EQ $34
3E-               1030 END    .EQ $3E,3F    ADDRESS OF END OF RANGE
40-               1040 PNTR   .EQ $40,41,42   START OF RANGE, CURRENT PNTR
                  1050 *--------------------------------
0200-             1060 WBUF   .EQ $200
0280-             1070 KBUF   .EQ $280
02C0-             1080 MBUF   .EQ $2C0
                  1090 *--------------------------------
FCCA-             1100 NXTCHR .EQ $FCCA
FD8E-             1110 CROUT  .EQ $FD8E
FDDA-             1120 PRBYTE .EQ $FDDA
FDED-             1130 COUT   .EQ $FDED
                  1140 *--------------------------------
```

```
                            .OP 8
                  T
000800- A9 13               LDA #SEARCH  Setup Control-Y vector
000802- 8F A1 00 E1         STA $E100A1
000806- A9 08               LDA /SEARCH
000808- 8F A2 00 E1         STA $E100A2
00080C- A9 00               LDA #0
00080E- 8F A3 00 E1         STA $E100A3
000812- 60                  RTS
                  *--------------------------------
                  SEARCH
000813- 08                  PHP          Save mode, get into emulation mode
000814- 38                  SEC
000815- FB                  XCE
000816- 08                  PHP
                  *--------------------------------
000817- 20 54 08            JSR GET.PATTERN   Crack the search pattern
00081A- B0 34               BCS .99           ...nothing there or too long
00081C- 8E 00 02            STX WBUF          save # bytes in pattern
00081F- AF 3E 01 E1         LDA $E1013E       Bank to search
000823- 85 42               STA PNTR+2        make 3-byte address
                  *---Compare pattern--------------
000825- A0 FF     .1        LDY #-1
000827- C8        .2        INY
000828- B7 40               LDA >(PNTR),Y     Byte from source
00082A- 59 80 02            EOR KBUF,Y        Byte from pattern
00082D- 39 C0 02            AND MBUF,Y        Byte from mask
000830- D0 0E               BNE .3            ...does not pass inspection
000832- CC 00 02            CPY WBUF          At end of pattern yet?
000835- 90 F0               BCC .2            ...no
                  *---Found a match----------------
000837- A5 42               LDA PNTR+2        Print 3-byte address
000839- A2 40               LDX #PNTR
00083B- A0 A0               LDY #" "          ...followed by a blank
00083D- 20 DF 08            JSR PRADR
                  *---Slip to next postion---------
000840- A5 40     .3        LDA PNTR     Set carry if at end of range
000842- C5 3E               CMP END
000844- A5 41               LDA PNTR+1
000846- E5 3F               SBC END+1
000848- E6 40               INC PNTR     Bump comparison pointer
00084A- D0 02               BNE .4
00084C- E6 41               INC PNTR+1
00084E- 90 D5     .4        BCC .1
                  *--------------------------------
000850- 28        .99       PLP          Restore machine mode
000851- FB                  XCE
000852- 28                  PLP
000853- 60                  RTS
                  *--------------------------------
                  GET.PATTERN
000854- A4 34               LDY YSAVE
000856- A2 FF               LDX #-1
000858- E8        .1        INX
000859- A9 FF               LDA #$FF     Assume mask =$FF
00085B- 9D C0 02  .2        STA MBUF,X   Store a mask value
00085E- A9 00               LDA #$00     Start Keybyte = $00
000860- 9D 80 02  .3        STA KBUF,X
000863- C0 7F               CPY #$7F
000865- B0 3D               BCS .7       ...LINE TOO LONG
000867- 20 CA FC            JSR NXTCHR   GET UPPER-CASE CHAR AT WBUF,Y
00086A- C9 A0               CMP #" "     Is it a space?
00086C- F0 EA               BEQ .1       ...yes, next byte
00086E- C9 BA               CMP #":"
000870- 90 11               BCC .5       < $BA, must be 0...9
000872- D0 05               BNE .4       > $BA, must be A...F
000874- BD 80 02            LDA KBUF,X   = $BA, ":", means to put
000877- B0 E2               BCS .2       ...ALWAYS  value into MASK
                  *---Try letter A...F-------------
000879- C9 C7     .4        CMP #"G"
00087B- B0 27               BCS .7       ...ERROR
00087D- C9 C1               CMP #"A"
00087F- 90 23               BCC .7       ...ERROR
000881- E9 07               SBC #7       "A"..."F"
                  *---Try hex digit range----------
000883- C9 B0     .5        CMP #"0"     Is it 0...9 or A...F?
000885- 90 14               BCC .6       ...no
```

```
000887- 29 0F                 AND #$0F      ...yes, isolate digit
000889- 1E 80 02              ASL KBUF,X    Shift previous value
00088C- 1E 80 02              ASL KBUF,X
00088F- 1E 80 02              ASL KBUF,X
000892- 1E 80 02              ASL KBUF,X
000895- 1D 80 02              ORA KBUF,X    Merge with digit
000898- 4C 60 08              JMP .3
                        *---End of line or error---------
00089B- C9 8D           .6    CMP #$8D
00089D- D0 05                 BNE .7        ...ERROR
00089F- 88                    DEY
0008A0- 84 34                 STY YSAVE
0008A2- 18                    CLC
0008A3- 60                    RTS
                        *---Bad char or line too long----
0008A4- 84 34           .7    STY YSAVE
0008A6- A9 8D                 LDA #$8D      STORE <RETURN> IN MONITOR
0008A8- 99 01 02              STA WBUF+1,Y       INPUT BUFFER
0008AB- A9 87                 LDA #$87      RING THE BELL
0008AD- 20 ED FD              JSR COUT
0008B0- 38                    SEC           INFORM OF ERROR
0008B1- 60                    RTS
                        *--------------------------------
                        DISPLAY.ADDRESSES
0008B2- 08                    PHP
0008B3- 38                    SEC
0008B4- FB                    XCE
0008B5- 08                    PHP
                        *--------------------------------
0008B6- A5 34                 LDA YSAVE        WBUF,Y POINTS TO NEXT CHAR
0008B8- 20 DA FD              JSR PRBYTE          AFTER THE CTRL-Y
0008BB- 20 8E FD              JSR CROUT
                        *--------------------------------
0008BE- AF 3F 01 E1           LDA $E1013F
0008C2- A2 42                 LDX #PNTR+2
0008C4- A0 BC                 LDY #"<"
0008C6- 20 DF 08              JSR PRADR
                        *--------------------------------
0008C9- AF 3E 01 E1           LDA $E1013E
0008CD- A2 40                 LDX #PNTR
0008CF- A0 AE                 LDY #"."
0008D1- 20 DF 08              JSR PRADR
                        *--------------------------------
0008D4- A2 3E                 LDX #END
0008D6- A0 8D                 LDY #$8D
0008D8- 20 E7 08              JSR PRADRO
                        *--------------------------------
0008DB- 28                    PLP
0008DC- FB                    XCE
0008DD- 28                    PLP
0008DE- 60                    RTS
                        *--------------------------------
0008DF- 20 DA FD        PRADR JSR PRBYTE     PRINT BANK
0008E2- A9 AF                 LDA #"/"
0008E4- 20 ED FD              JSR COUT
                        PRADRO
0008E7- B5 01                 LDA 1,X       PRINT HI-BYTE OF ADDRESS
0008E9- 20 DA FD              JSR PRBYTE
0008EC- B5 00                 LDA 0,X       PRINT LO-BYTE OF ADDRESS
0008EE- 20 DA FD              JSR PRBYTE
0008F1- 98                    TYA           PRINT TRAILING CHAR
0008F2- 4C ED FD              JMP COUT
                        *--------------------------------
```

Another project I thought of was modifying this search program so that it could run in an older Apple. Naturally we would lose the 3-byte addresses, so the program actually becomes simpler. A listing of this simpler version is shown below.

Æ Update . . .

The fastest IIGS memory product on the market is now available with Applied Engineering's GS-RAM and GS-RAM Plus memory cards. AE's GS-RAM and GS-RAM Plus now include a new disk caching feature which can be used independently or with the Apple IIGS RAMdisk. (Although a RAMdisk is faster, disk caching is easier to use.) Caching significantly improves Apple Disk 3.5 access time. The new caching technique uses highly efficient and optimized machine language programs providing ultra fast, state-of-the-art dynamic disk caching in both ProDOS 8 and ProDOS 16 environments. This new enhancement makes GS-RAM and GS-RAM Plus the fastest IIGS memory cards on the market today. Ultra fast disk caching is now included with all GS-RAM and GS-RAM Plus purchases. Current GS-RAM users can obtain an upgrade for $10.

Applied Engineering's DataLink modem is now ready to order with delivery in 2-3 weeks. DataLink is the newest and most advanced internal 1200/300 baud modem available today. DataLink incorporates the latest "modem on a chip" technology. Some of the components are so advanced they did not exist only one year ago. Because slot 2 is the normal location for a modem, Applied Engineering made an extra effort to make DataLink the only internal modem small enough to fit in slot 2 without interfering with the Apple IIGS's built-in fan. DataLink works in the Apple IIe and II+ as well. DataLink includes powerful communications software both in EPROM and on disk. This state-of-the-art modem has a retail price of only $219 and is covered by a 5 year warranty.

New video digitizer is in the final stages of development. Unlike older techniques, Applied Engineering's video digitizer incorporates its own high speed memory which solves many of the problems that the current DMA dependent video digitizers inherently have. Whereas other video digitizers are IIGS only, this new process will allow the Applied Engineering card to also work in an Apple IIe and II+ as well as the IIGS. Video digitizers on the market at present require many video frames in order to digitize a picture. This results in a digitized image that shows tearing or distortion if there is motion when the image is digitized. Applied Engineering's design is capable of digitizing a complete color or black-and-white image in a single frame, thereby eliminating distortion. Because the design incorporates three separate "flash analog-to-digital converters" it can actually digitize to a greater resolution than the Apple is capable of displaying on screen. However, the high resolution image can be reproduced on most printers and the video image is displayed to the maximum resolution on the Apple's monitor. The product should be available in September. A price has not been set but is expected to be in the $250 range.

Nonvolatile GS memory board soon to be released by Applied Engineering. The new card can piggyback to the GS-RAM and GS-RAM Plus cards or be used independently and uses standard dynamic RAM chips. It is available in 1 megabyte and 2 megabyte sizes. The memory on the board can be used as system memory or partitioned as a ROMdisk in 128K increments. If a RamCharger battery back up unit is connected, the memory is non-volatile. There are other techniques for creating non-volatile memory in the Apple IIGS memory expansion slot. One is the use of EPROMS. The engineers at Applied rejected this approach because of the difficulty in programming and reprogramming whenever customers wanted to update their software. The difficulty in programming and changing EPROMs combined with their relatively small memory capacities meant this approach was not acceptable. Another technique is to use SRAMs (Static RAMs) or EEPROMs (Electrically Erasable PROMs). These memory devices are easily programmed by the user but present the disadvantages of relatively small capacity and high cost. AE's solution uses 1 megabyte DRAMs with a battery backed-up refresh circuit. This allows the currently available RamCharger battery back-up unit to be plugged directly into the new expander to give permanent non-volatile storage. The card can also be used without the RamCharger as additional memory expansion. AE's solution has the advantages of large capacity and low cost.

Applied Engineering's new MS-DOS card (code named "Little Blue") is proceeding as planned. AE expects to be shipping product in October or November. Testing indicates that the product will be compatible with approximately 98% of IBM software; all major MS-DOS software is compatible. The card is expected to retail for approximately $500.

Applied Engineering is seeking an experienced 6502 machine language programmer to develop software and firmware for its future line of Apple peripherals. Minimum 2 years Apple programming experience required. Send your resume to Applied Engineering, P.O. Box 798, Carrollton, TX 75006 Attn: Personnel.

```
                 *SAVE S.SEARCH.II
                 *-------------------------------
34-              YSAVE  .EQ $34
3E-              END    .EQ $3E,3F   ADDRESS OF END OF RANGE
40-              PNTR   .EQ $40,41,42   START OF RANGE, CURRENT PNTR
                 *-------------------------------
0200-            WBUF   .EQ $200
0280-            KBUF   .EQ $280
02C0-            MBUF   .EQ $2C0
                 *-------------------------------
FD8E-            CROUT  .EQ $FD8E
FDDA-            PRBYTE .EQ $FDDA
FDED-            COUT   .EQ $FDED
                 *-------------------------------
                 T
0800- A9 0B             LDA #SEARCH  Setup Control-Y vector
0802- 8D F9 03          STA $3F9
0805- A9 08             LDA /SEARCH
0807- 8D FA 03          STA $3FA
080A- 60                RTS
                 *-------------------------------
                 SEARCH
080B- 20 45 08          JSR GET.PATTERN   Crack the search pattern
080E- B0 34             BCS .99           ...nothing there or too long
0810- 8E 00 02          STX WBUF          save # bytes in pattern
                 *---Compare pattern--------------
0813- A0 FF      .1     LDY #-1
0815- C8         .2     INY
0816- B1 40             LDA (PNTR),Y      Byte from source
0818- 59 80 02          EOR KBUF,Y        Byte from pattern
081B- 39 C0 02          AND MBUF,Y        Byte from mask
081E- D0 14             BNE .3            ...does not pass inspection
0820- CC 00 02          CPY WBUF          At end of pattern yet?
0823- 90 F0             BCC .2            ...no
                 *---Found a match----------------
0825- A5 41             LDA PNTR+1
0827- 20 DA FD          JSR PRBYTE
082A- A5 40             LDA PNTR
082C- 20 DA FD          JSR PRBYTE
082F- A9 A0             LDA #" "
0831- 20 ED FD          JSR COUT
                 *---Slip to next postion---------
0834- A5 40      .3     LDA PNTR     Set carry if at end of range
0836- C5 3E             CMP END
0838- A5 41             LDA PNTR+1
083A- E5 3F             SBC END+1
083C- E6 40             INC PNTR     Bump comparison pointer
083E- D0 02             BNE .4
0840- E6 41             INC PNTR+1
0842- 90 CF      .4     BCC .1
                 *-------------------------------
0844- 60         .99    RTS
                 *-------------------------------
                 GET.PATTERN
0845- A4 34             LDY YSAVE
0847- A2 FF             LDX #-1
0849- E8         .1     INX
084A- A9 FF             LDA #$FF     Assume mask =$FF
084C- 9D C0 02   .2     STA MBUF,X   Store a mask value
084F- A9 00             LDA #$00     Start Keybyte = $00
0851- 9D 80 02   .3     STA KBUF,X
0854- C0 7F             CPY #$7F
0856- B0 3D             BCS .7       ...LINE TOO LONG
0858- 20 A3 08          JSR NXTCHR   GET UPPER-CASE CHAR AT WBUF,Y
085B- C9 A0             CMP #" "     Is it a space?
085D- F0 EA             BEQ .1       ...yes, next byte
085F- C9 BA             CMP #":"
0861- 90 11             BCC .5       < $BA, must be 0...9
0863- D0 05             BNE .4       > $BA, must be A...F
0865- BD 80 02          LDA KBUF,X   = $BA, ":", means to put
0868- B0 E2             BCS .2       ...ALWAYS   value into MASK
                 *---Try letter A...F-------------
086A- C9 C7      .4     CMP #"G"
086C- B0 27             BCS .7       ...ERROR
086E- C9 C1             CMP #"A"
0870- 90 23             BCC .7       ...ERROR
0872- E9 07             SBC #7       "A"..."F"
```

```
                  1770 *---Try hex digit range----------
0874- C9 B0       1780 .5     CMP #"0"      Is it 0...9 or A...F?
0876- 90 14       1790        BCC .6        ...no
0878- 29 0F       1800        AND #$0F      ...yes, isolate digit
087A- 1E 80 02    1810        ASL KBUF,X    Shift previous value
087D- 1E 80 02    1820        ASL KBUF,X
0880- 1E 80 02    1830        ASL KBUF,X
0883- 1E 80 02    1840        ASL KBUF,X
0886- 1D 80 02    1850        ORA KBUF,X    Merge with digit
0889- 4C 51 08    1860        JMP .3
                  1870 *---End of line or error---------
088C- C9 8D       1880 .6     CMP #$8D
088E- D0 05       1890        BNE .7        ...ERROR
0890- 88          1900        DEY
0891- 84 34       1910        STY YSAVE
0893- 18          1920        CLC
0894- 60          1930        RTS
                  1940 *---Bad char or line too long----
0895- 84 34       1950 .7     STY YSAVE
0897- A9 8D       1960        LDA #$8D      STORE <RETURN> IN MONITOR
0899- 99 01 02    1970        STA WBUF+1,Y      INPUT BUFFER
089C- A9 87       1980        LDA #$87      RING THE BELL
089E- 20 ED FD    1990        JSR COUT
08A1- 38          2000        SEC           INFORM OF ERROR
08A2- 60          2010        RTS
                  2020 *--------------------------------
08A3- B9 00 02    2030 NXTCHR LDA WBUF,Y
08A6- C8          2040        INY
08A7- C9 E1       2050        CMP #$E1
08A9- 90 06       2060        BCC .1
08AB- C9 FB       2070        CMP #$FB
08AD- B0 02       2080        BCS .1
08AF- 29 DF       2090        AND #$DF
08B1- 60          2100 .1     RTS
                  2110 *--------------------------------
```

Reading the IIgs ROMs....................Bob Sander-Cederlof

A few days ago a friend and I were burning some EPROMs using the SCRG PromGramer in my IIgs. The friend said, "Why don't we use this card to read the IIgs ROM?" "We could put the PromGramer in another machine, and then put the IIgs ROMs in it and read them."

Well, I don't really know why he wanted to do this. He has his own IIgs already anyway. Nevertheless, I can think of reasons to want to capture all of the ROM info on disk files. Disassembly, for one, using the S-C DisAssembler. Another reason would be to be able to compare different machines to see if they have the same ROM contents. But do we have to take out the chips to read them?

No. The following simple commands, issued from inside the S-C Macro Assembler running in your IIgs, will write the entire ROM contents to a series of eight files. It works under either ProDOS or DOS 3.3:

```
$00/1000<FE/0000.3FFFM
BSAVE ROM.FE.0,A$1000,L$4000
$00/1000<FE/4000.7FFFM
BSAVE ROM.FE.4,A$1000,L$4000
$00/1000<FE/8000.BFFFM
BSAVE ROM.FE.8,A$1000,L$4000
$00/1000<FE/C000.FFFFM
BSAVE ROM.FE.C,A$1000,L$4000
$00/1000<FF/0000.3FFFM
BSAVE ROM.FF.0,A$1000,L$4000
$00/1000<FF/4000.7FFFM
BSAVE ROM.FF.4,A$1000,L$4000
$00/1000<FF/8000.BFFFM
BSAVE ROM.FF.8,A$1000,L$4000
$00/1000<FF/C000.FFFFM
BSAVE ROM.FF.C,A$1000,L$4000
```

I found it convenient to put those commands in a text file, which I named GET.ROMS. Then, whenever I want to use it, I can just type EXEC GET.ROMS. I took my disk down to the local Heathkit store around 10:45 this morning and came back with a copy of their ROM.

It strikes me just now that I am being inconsistent. Sometimes I am saying ROM, and sometimes ROMs. There is a reason....

Real IIgs machines come with one system ROM chip, which is a 1 megabit ROM. It holds all 128K bytes, in only one 28-pin package. I presume it is uses pinouts like a 27512 EPROM, but selects one half or the other of the 1024K bits using what would normally be the programming pin on the 27512. My IIgs, on the other hand, is a prototype machine. It has an adapter plugged into the single ROM socket, which holds four 27256 EPROM chips and one 74F139. They call it the "airplane ROMs". I like it, because if I want to I can easily burn a new version of the firmware and plug it in. Some enterprising soul might find a market for a board like this.

More About Patching Apple's ProDOS Releases

You remember in our March issue we talked about patches to fix Version 1.3 of ProDOS. Apple has pulled this version off the market, but there are still a lot of copies floating around. The patches we gave in the March article should make ProDOS 1.3 as good as any other version, but who knows?

Anyway, we heard through the grapevine that some unofficial copies of version 1.4 are out, and that a brand new bug has surfaced in this one. It seems someone put a "LDA $C09C,X" where "LDA $C08E,X" should be.

I ran across a listing in the Washington Apple Pi newsletter (May 1987 issue, page 16) of an Applesoft program which can install all known necessary patches in versions 1.1.1, 1.2, 1.3, and 1.4 of ProDOS. The program was originally written by Stephen Thomas to fix version 1.1.1, when the problem of the four STA's to the stepper motor soft-switches was discovered. (See Nov 86 AAL) Later Glen Bredon modified it to make the corresponding patches to later versions, as well as to fix the additional new bugs. I have further modified it, in an attempt to make it easier to understand.

```
100  TEXT : HOME :E = 0: PRINT "PRODOS PATCH PROGRAM"
110  IF  PEEK (116) < 128 THEN E = 1: GOTO 900: REM ENUF MEM?
120  ONERR  GOTO 900
130  REM ---READ PRODOS FILE---
140  PRINT  CHR$ (4)"UNLOCK PRODOS"
150  PRINT  CHR$ (4)"BLOAD PRODOS,TSYS,A$2000"
200  REM ---SEARCH $4000-$60FF FOR PATTERN---
210 V = 1: FOR B = 64 TO 96:A = B * 256
220  IF  PEEK (A + 4) <  > 189 THEN 250
230  IF  PEEK (A + 5) <  > 156 THEN 290
240  IF  PEEK (A + 6) = 192 THEN V = 3:B = 96: GOTO 290: REM VERSION 1.4
250  IF  PEEK (A + 4) <  > 234 THEN 290
260  IF  PEEK (A + 5) <  > 234 OR  PEEK (A + 6) <  > 234 THEN 290
270  IF  PEEK (A + 7) <  > 234 OR  PEEK (A + 8) <  > 234 THEN 290
280 V = 2:B = 96: REM VERSION BEFORE 1.4
290  NEXT B:E = 2: ON V GOTO 900,300,700
300  REM ---FOUND VERSION BEFORE 1.4---
310  POKE A + 4,189: POKE A + 5,142: POKE A + 6,192: REM "LDA $C08E,X"
400  REM ---LOOK FOR OTHER PATCH AREA---
410 A =  PEEK (A + 2) + 256 *  PEEK (A + 3) - 13 * 4096 + A + 5
420 E = 3: IF A < 4 * 4096 OR A > 6 * 4096 THEN 900
430  IF  PEEK (A) <  > 157 OR  PEEK (A + 1) <  > 157 THEN 500
440  IF  PEEK (A + 2) <  > 157 OR  PEEK (A + 3) <  > 157 THEN 500
450  REM ---FOUND VERSION 1.1.1 OR 1.2, SO CHANGE "LDA" TO "STA"---
460  FOR I = 0 TO 9 STEP 3: POKE A + I,189: NEXT I
470 V$ = "1.1.1": GOTO 800
500  REM  ---VERSION 1.3---
510  FOR I = 0 TO 12: READ B: IF  PEEK (A + I) <  > B THEN  GOTO 900
520  NEXT I: DATA 160,8,189,128,192,232,232,136,208,248,234,234,96
530  FOR I = 0 TO 11: READ B: POKE A + I,B: NEXT I
540  DATA 189,128,192, 189,130,192, 189,132,192, 189,134,192
550 A = 4 * 4096 + 12 * 256 + 12 * 16 + 13: REM  ADDRESS = $4CCD
560  FOR I = 0 TO 3: READ B: IF  PEEK (A + I) <  > B THEN 900
570  NEXT I: POKE A,240: REM CHANGE "BRA" TO "BEQ"
580 V$ = "1.3": GOTO 800
590  DATA 128,6,190,0
```

```
700  REM ---VERSION 1.4---
710  POKE A + 5,142: REM "LDA $C09C,X" TO "LDA $C08E,X"
720 V$ = "1.4"
800  REM ---WRITE PATCHED VERSION ON DISK---
810  PRINT  CHR$ (4)"BSAVE PRODOS,TSYS,A$2000"
820  PRINT  CHR$ (4)"LOCK PRODOS"
830  PRINT "PATCHES COMPLETED TO VERSION "V$: END
900  REM ---ERROR HANDLER---
910  PRINT  CHR$ (7)"ERROR!  NO PATCHES WERE MADE."
915  ON E GOTO 930,940,950
920  PRINT "PRODOS FILE NOT FOUND.": END
930  PRINT "NOT ENOUUGH ROOM TO LOAD PRODOS.": END
940  PRINT "PATCH LOCATION NOT FOUND.": END
950  PRINT "PRODOS FILE MAY HAVE BEEN PATCHED,"
960  PRINT "ALREADY, OR IS NOT A COMPATIBLE VERSION."
970  END
```

Lines 100-150 read the ProDOS system file into memory. Then Lines 200-290 search every page from $4000 through $60FF for either five NOPs starting at $xx04 or a "LDA $C09C,X" instruction at $xx04. If neither is found, nothing is patched. If the five NOPs are found, we have version 1.1.1, 1.2, or 1.3. If the LDA is found, we have version 1.4. If it is version 1.4, the only patch needed is to change it to "LDA $C08E,X", which is done at lines 700-720.

Older versions all need "LDA $C08E,X" poked where the five NOPs were, so line 310 takes care of this. Then we look at the address in the operand field of the instruction just prior to the five NOPs. This is a JSR to a little subroutine which we need to modify. Line 410 computes the location within the system file image for the twelve bytes we need to change.

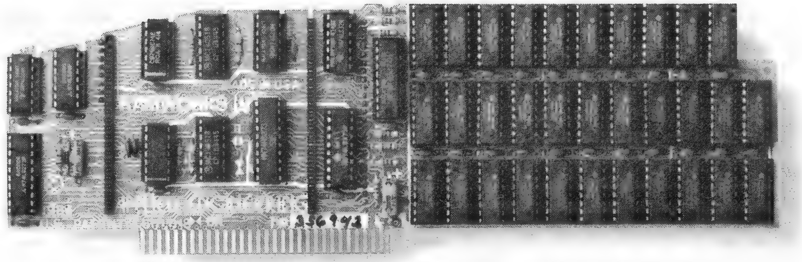
There are several possible versions of this subroutine. If it is a series of "STA $C08x,X" instructions, we have version 1.1.1 or 1.2 and the STA opcodes should be changed to LDA opcodes. Lines 430 and 440 test for STA opcodes, and lines 450-470 make the changes. On the other hand, if the subroutine is like Apple put in version 1.3 we will replace it with a series of four LDAs just like we made in the older versions. Lines 500-590 handle this, and also change an errant "BRA" opcode to a "BEQ" opcode.

Finally, lines 800-830 write out the modified code and re-LOCK the file. I would be careful to check the changes made before doing this to every copy I own, if I were you. And bear in mind that Apple as a company has never authorized any of these changes. (They have only made them necessary, by their own incorrect changes.)

While this article was waiting for the press, Apple finally sent out correct copies of version 1.4. I received my master copy June 1st, and checked it against our patched version 1.3. They were identical except for the copyright dates and version numbers. The official date on this GOOD version 1.4 is April 17, 1987.

RamWorks® III

Patented Performance from the Recognized Leader



With battery backed RAM port, RGB port, increased memory capacity, full software compatibility and more compact design, RamWorks III is a generation ahead.

RamWorks III is the newest 3rd generation RAM card for the Apple IIe. It incorporates all of the technology and improvements that years of experience and over a hundred thousand sales have given us. By selling more memory cards than anyone else and listening to our customers, we were able to design a memory card that has the ultimate in performance, quality, compatibility and ease of use. A design so advanced it's patented. We call it RamWorks III, you'll call it awesome!

The AppleWorks Amplifier.

While RamWorks III is recognized by all memory intensive programs, NO other expansion card comes close to offering the multitude of enhancements to AppleWorks that RamWorks III does. Naturally, you'd expect RamWorks III to expand the available desktop, after all Applied Engineering was a year ahead of everyone else *including Apple* in offering more than 55K, and we still provide the largest AppleWorks desktops available. But a larger desktop is just part of the story. Look at all the AppleWorks enhancements that even Apple's own card does not provide and *only* RamWorks III does. With a 256K or larger RamWorks III, *all* of AppleWorks (including printer routines) will automatically load itself into RAM dramatically increasing speed by eliminating the time required to access the program disk drive. Switch from word processing to spreadsheet to database at the speed of light with no wear on disk drives.

Only RamWorks eliminates AppleWorks' internal memory limits, increasing the maximum number of records available from 1,350 to over 25,000. *Only* RamWorks increases the number of lines permitted in the word processing mode from 2,250 to over 15,000. And *only* RamWorks offers a built-in printer buffer, so you won't have to wait for your printer to stop before returning to AppleWorks. RamWorks even expands the clipboard. And auto segments large files so they can be saved on two or more disks. You can even have Pinpoint or MacroWorks and your favorite spelling checker in RAM for instant response.

RamWorks, nothing comes close to enhancing AppleWorks so much.

The Most Friendly, Most Compatible Card Available.

Using RamWorks III couldn't be easier because it's compatible with more off-the-shelf software than any other RAM card. Popular programs like AppleWorks, Pinpoint, Catalyst, MouseDesk, HowardSoft, FlashCalc, Pro-Filer, Managing Your Money, SuperCalc 3a, and MagiCalc to name a few (and *all* hardware add on's like ProFile and Sider hard disks). RamWorks is even compatible with software written for Apple cards. But unlike other cards, RamWorks plugs into the IIe auxiliary slot providing our super sharp 80 column text (U.S. Patent #4601018) in a completely integrated system while leaving expansion slots 1 through 7 available for other peripheral cards.

RamWorks III is compatible with all Apple IIe's, enhanced, unenhanced, American or European versions.

Highest Memory Expansion.

Applied Engineering has always offered the largest memory for the IIe and RamWorks III continues that tradition by expanding to 1 full MEG on the main card using standard RAMs, more than most will ever need (1 meg is about 500 pages of text)...but if you do ever need more than 1 MEG, RamWorks III has the widest selection of expander cards available. Additional 512K, 2 MEG, or 16 MEG cards just snap directly onto RamWorks III by plugging into the industry's only low profile (no slot 1 interference) fully decoded memory expansion connector. You can also choose non-volatile, power independent expanders allowing permanent storage for up to 20 years.

It Even Corrects Mistakes.

If you've got some other RAM card that's not being recognized by your programs, and you want RamWorks III, you're in luck. Because all you have to do is plug the memory chips from your current card into the expansion sockets on RamWorks to recapture most of your investment!

The Ultimate in RGB Color.

RGB color is an option on RamWorks and with good reason. Some others combine RGB color output with their memory cards, but that's unfair for those who don't need RGB *and* for those that do. Because if you don't need RGB

ABC*DE=FG*HI Puzzle Solved...............Bob Sander-Cederlof

A+ Magazine has an interesting puzzle each month, with a challenge to solve it with your Apple and possibly win a prize. The June 1987 puzzle (see page 110, "Computer Calisthenics" by Michael Wiesenberg) involves writing a program to find all possible solutions to the equation ABC*DE=FG*HI, where each letter represents a different digit between 1 and 9. All nine digits are different.

I wrote a quick-and-dirty program in Applesoft, and then refined it a little for speed. Maybe speed isn't the correct work, because even the refined result takes nearly six minutes to find the eleven solutions. The first solution my program found was the same as the one example solution given in the A+ article: 158*23 = 46*79.

Puzzles of this type can be solved, if you have enough time, by trying all possible combinations. A series of nested FOR loops, one for each digit, will produce all possible nine-digit arrangements. Appropriate tests can weed out all values which have more than one letter sharing the same digit. Inside all the loops you test the letter assignments against the puzzle equation, and print the solution if it passes.

This will take way too long, even on a computer, so you start looking for ways to eliminate some combinations. The first thing I noticed is that I can eliminate symmetric solutions by forcing HI>FG. This means H>F, so I can run my FOR loop for H from F+1 to 9, instead of from 1 to 9.

The smallest possible value for abc*de is 245*13=3185. This means F cannot be 1 or 2, because even 29*99 is too small. A little more examination shows that FG must be at least 34, so I put this into my program.

Neither C nor E can be 5, because this would force either G or I to also be 5. I also notice that you cannot have a 1 on the right-hand side of the puzzle equation. If there were a 1 there, the largest possible FG*HI would be 91*87 = 7917. This is still smaller than the smallest possible left-hand side without a 1 (356*24 = 8544), so it cannot work.

Similar analysis shows that D cannot be 8 or 9. The largest possible FG*HI is 87*96 = 8352. If D is 8, the smallest ABC*DE is 134*82 = 10988; D=9 is even worse. This also cannot work, so I limit my D-loop to values from 1 to 7.

I also refined the program in the area of testing whether a digit has already been used by a previous letter. I maintain an array of flags. At the top of each loop I test the flag array entry to see if the digit has already been used. If not, I mark it used and continue. If it is already in use, I skip around all the inner loops to the corresponding NEXT. Here is the Applesoft version of my program:

```
10  DIM N(9): FOR I = 1 TO 9:N(I) = I: NEXT :EPS = .0001
100  FOR F = 3 TO 8:N(F) = 0
110  FOR GX = 2 TO 9:G = N(GX): IF G = 5 OR G = 0 THEN 320
115 FG = F * 10 + G: IF FG < 34 THEN 320
120 N(GX) = 0: FOR HX = F + 1 TO 9:H = N(HX): IF H = 0 THEN 310
130 N(HX) = 0: FOR IX = 2 TO 9:I = N(IX): IF I = 0 OR I = 5 THEN 300
140 HI = H * 10 + I:P = FG * HI: IF P < 3240 THEN 300
150 N(IX) = 0: FOR DX = 1 TO 8:D = N(DX): IF D = 0 THEN 290
160 N(DX) = 0: FOR EX = 1 TO 9:E = N(EX): IF E = 0 OR E = 5 THEN 280
170 DE = D * 10 + E:Q = P / DE: IF Q < 123 OR Q <  >  INT (Q + EPS) THEN 280
180 N(EX) = 0:A =  INT (Q / 100 + EPS): IF N(A) = 0 THEN 270
190 N(A) = 0:B =  INT (Q / 10 - A * 10 + EPS): IF N(B) = 0 THEN 260
200 N(B) = 0:C =  INT (Q - A * 100 - B * 10 + EPS): IF N(C) = 0 THEN 250
210 N = N + 1: PRINT  SPC( N < 10);N":  "FG" X "HI" = "DE" X "Q" = "P
250 N(B) = B
260 N(A) = A
270 N(EX) = EX
280  NEXT EX:N(DX) = DX
290  NEXT DX:N(IX) = IX
300  NEXT IX:N(HX) = HX
310  NEXT HX:N(GX) = GX
320  NEXT GX:N(F) = F: NEXT F
```

I wanted to see how hard it would be to re-write the above program in assembly language, and if so how much speedier it would be. I am not proud that it took me over four hours to perfect the assembly language version. But the result is nice. It executes in less than seven seconds! This does not necessarily argue well for assembly language programming, if all I care about is the answers to the puzzle. But if I view it as an example, and consider that the same speedup may be possible in much larger and much more frequently used programs, it does make assembly language look good. That is why programs like the S-C Macro Assembler and even the Applesoft firmware itself are written in assembly language.

I used the same overall approach in assembly language version. Lines 1040-1090 initialize an array of flags I use to quickly check whether a digit is already in use. The flag values are either 0 or 1: 0 means a digit is in use, and 1 means it is available. The array is accessed by using the digit value for an index. I can both test a flag and change it to zero with one instruction: LSR FLAGS,X will shift bit 0 into carry. If the flag was 0, it still is and carry is clear. If the flag was 1 it changes to 0 and carry is set.

Lines 1110-1520 are a equivalent of FOR statements, starting up a series of six nested loops to generate values for D, E, F, G, H, and I. Lines 1580-1880 are the equivalent to the NEXT statements. All the flag array handling is also included in these two groups of lines. I put the prior knowledge about the ranges of possible values for these six letters into the FOR-loop values, and eliminate the value 5 for letters E, G, and I.

Lines 1540-1560 call subroutines to test the choices for letters D through I and print the resulting equation if it is a valid solution to the puzzle. Both of these subroutines use some really interesting techniques.

The PRINT subroutine (lines 2820-3060) is controlled by a

format string in lines 3080-3160. This string has two kinds of bytes: index values between 0 and 8, and ASCII characters between 80 and FF. The index values point to the table of digits A through I. The print loop, lines 2840-2910, only needs four extra lines to pick up digits out of the digit table and convert them to ASCII. Line 2850 branches directly to the JSR COUT in line 2890 if the format byte is an ASCII character already. Otherwise, lines 2860-2880 use the format byte as an index to pick up the digit and merge $B0 with it to convert it to ASCII. This is not only neat, it is also short and fast. Lines 2940-3050 allow you to pause and abort the program by typing any key to pause, <RETURN> to abort.

The COMPUTE.ABC subroutine (lines 1920-2710) checks the chosen values for D through I to see if they are a valid solution to the puzzle. If they are, values for A, B, and C will be chosen in the process of the check. Then the subroutine returns with carry clear to indicate a valid solution. If the solution is not valid, the subroutine will return with carry set.

Lines 1930-2010 call on CALC.XX (lines 2730-2800) to calculate binary values for DE, FG, and HI. I could have added a test here to be sure that DE>33, but it did not seem to be worth the effort. I didn't try it, though, so I might be wrong. CALC.XX mutiplies the first of each pair of digits by ten, and then adds the second.

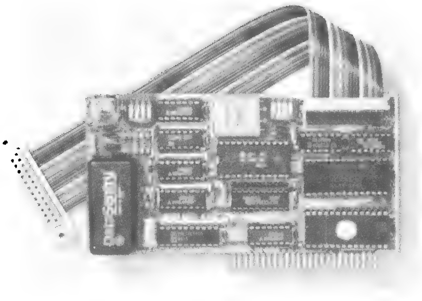
Lines 2020-2160 compute WXYZ = FG*HI. The highest possible value for FG will be 79 and for HI will be 98. Since we are limiting F to the range 1 through 7, the largest possible combination will be 78*96 = 7488. This is $1D40. The calculation is a simple 8-bit by 8-bit multiplication, with a 16-bit result.

Lines 2170-2330 compute ABC = (FG*HI)/DE. I could have used two or three more nested loops to pick values for A, B, and C, but "it seemed like a good idea at the time" to do it this way. Now I think two more loops to pick A and B using the same techniques as for the other letters, plus a simple search through the flags to pick C, would be nicer. You might try it that way and compare the speeds of the two approaches. My calculation is a simple division with a 16-bit dividend, 8-bit divisor, 16-bit quotient, and 8-bit remainder. If the remainder is non-zero, then the numbers picked do not form a valid solution.

Lines 2340-2460 determine the value of A by essentially dividing ABC by 100. ABC is a 16-bit value, so the subtraction loop has to do a 16-bit subtraction. The loop subtracts one extra time, so line 2440 corrects the remainder (which will be the value BC). Lines 2450-2460 check to see if the value for A was already used for D through I.

Lines 2480-2580 do a similar operation to separate out the value of B, and by default leave the value of C in the A-register. Lines 2540-2570 check to see if the value for B was already used for D through I or for A. Finally, lines 2590-2660 check the value for C against all the other choices.

Serial Pro®
The intelligent multifunction card from Applied Engineering.



Serial Pro is a powerful multifunction card for the Apple IIe, II+, IIGS and compatible computers. Serial Pro combines a powerful serial port for interfacing to a printer, modem or other serial device, with a full function clock/calendar. Serial Pro's serial port is compatible with virtually all letter quality and dot matrix printers, including Apple's Imagewriter and Imagewriter II, and can communicate with the fastest external modems at baud rates from 50 to 19,200.

Set up is easy, because unlike other cards, there are no switches to program serial parameters or disks to use. Serial Pro has a built-in setup screen that allows all parameters to be set. And on-screen help that makes it super-easy. An on-board battery stores your selections for over 20 years, but you can change them at any time. Serial Pro is the perfect upgrade for Super Serial card owners looking for more performance or a built-in clock.

Serial Port Features • Fully Super Serial Card compatible but more versatile and easier to use • Compatible with printers • Compatible with modems • Prints graphics to printer (without software) in normal mode, 90° rotation, double size, side by side, and black/white inversion • Variable word length, number of stop bits and parity selection • Crystal controlled circuits are used to provide ultra accurate baud rates.

Clock Calendar Features • ProDos, DOS and AppleWorks compatible. Displays time and date on the AppleWorks screen • Automatically time and date stamps AppleWorks files • 20 year auto recharging nickel-cadmium battery • Realtime interrupts • Built-in on screen time and date settings (no software needed) • Does not use phantom slots.

Serial Pro is perfect for Apple IIe, II+, IIGS or compatible owners in need of a serial port or clock/calendar or both. But unlike other multifunction cards on the market, Serial Pro does not use phantom slots, so all the slots in your computer are useable.

Serial Pro comes complete with manual, cable and instructions for connecting to all the leading printers and modems.

Serial Pro **$159**

Order Today! See your dealer or call Applied Engineering, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

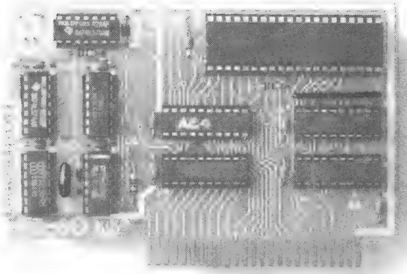
ÆAPPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

With the Z-80 Plus®, run over 5000 new CP/M® programs.



Now, get two computers in one, and all the advantages of both.

Enter the CP/M world with the new Z-80 Plus card from Applied Engineering, and introduce your Apple IIe, IIGS or II+ to thousands of CP/M programs. Only the Z-80 Plus comes standard with the new 5.1 software, the most advanced system ever for running CP/M programs.

The new 5.1 boasts advanced features like built-in disk emulation for popular memory expansion boards, boosting both system speed and storage capacity. And menu-driven utilities that let you get to work faster. The Z-80 Plus also lets you run older CP/M programs—all the way down to Version 1.6 (2.2 is the most popular).

The Z-80 Plus is the only card on the market capable of accessing more than 64K. If you have an extended 80-column card, all 128K is usable. And if you have RamWorks, RamFactor, GSRam, GSRam Plus, or an Apple memory card in your IIe, IIGS or II+, up to 1088K is available.

Each Z-80 Plus comes with our CP/M Ram Drive software, enabling most memory cards to work as a high-speed Ram disk so CP/M software can run up to *twenty times faster.* So packages like WordStar, dBASE II, Turbo Pascal and most others run at blinding speed.

Simply plug the Z-80 Plus into any slot in your Apple. You'll get the benefits of two computers in one—all at an unbelievably low price (only $149!).

• Fully compatible with ALL CP/M software • Fully compatible with 5¼" *and* 3½" disk drives as well as most hard disks including the Sider • Fully compatible with Microsoft disks (no pre-boot required) • Specifically designed for high speed operation • Runs WordStar, dBASE II, Turbo Pascal, and ALL other CP/M software with *no pre-boot.* • Semi-custom I.C. and low parts count allows Z-80 Plus to fly through CP/M programs with extremely low power consumption (we use the Z-80B) • Does EVERYTHING other Z-80 boards do, *plus* Z-80 interrupts • Five-year warranty

To get your Z-80 Plus or for more information see your dealer, or call 214-241-6060 9 a.m. to 11 p.m. seven days, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

ÆApplied Engineering™
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

CP/M is a registered trademark of Digital Research

As a final product, a program like this has little value (unless you win the contest!). However, it can be a great tool for perfecting your skill as a programmer. It is also a pleasant and harmless form of recreation. If you enjoy articles like these, let us know: we may do some more. Or, how about sending us one yourself?

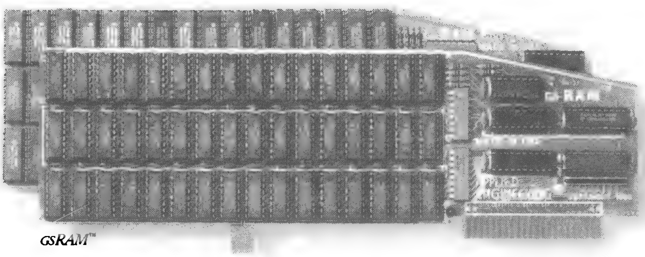
```
                *SAVE S.PUZZLE
                *--------------------------------
FDED-           COUT    .EQ $FDED
                *--------------------------------
                T
0800- A2 09             LDX #9
0802- A9 01             LDA #1          STORE 1 IN EACH FLAG
0804- 9D AC 09  .1      STA FLAGS,X
0807- CA                DEX
0808- D0 FA             BNE .1
                *--------------------------------
080A- A2 02             LDX #2          FOR F=3 TO 8
080C- E8        LOOP.F  INX
080D- 8E A0 09          STX F
0810- 5E AC 09          LSR FLAGS,X     mark digit in use
                *--------------------------------
0813- E0 03             CPX #3          IF F=3, START G-LOOP AT 4
0815- F0 02             BEQ LOOP.G
0817- A2 01             LDX #1          FOR G=2 TO 9
0819- E8        LOOP.G  INX
081A- 5E AC 09          LSR FLAGS,X
081D- 90 72             BCC NEXT.G      ...DIGIT ALREADY USED
081F- 8E A1 09          STX G
0822- E0 05             CPX #5
0824- F0 65             BEQ FIX.G       G can't be 5
                *--------------------------------
0826- AE A0 09          LDX F           FOR H=F+1 TO 9
0829- E8        LOOP.H  INX
082A- 5E AC 09          LSR FLAGS,X
082D- 90 58             BCC NEXT.H      ...DIGIT ALREADY USED
082F- 8E A2 09          STX H
                *--------------------------------
0832- A2 01             LDX #1          FOR I=2 TO 9
0834- E8        LOOP.I  INX
0835- 5E AC 09          LSR FLAGS,X
0838- 90 43             BCC NEXT.I      ...DIGIT ALREADY USED
083A- 8E A3 09          STX I
083D- E0 05             CPX #5
083F- F0 36             BEQ FIX.I       I can't be 5
                *--------------------------------
0841- A2 00             LDX #0          FOR D=1 TO 7
0843- E8        LOOP.D  INX
0844- 5E AC 09          LSR FLAGS,X
0847- 90 2A             BCC NEXT.D      ...DIGIT ALREADY USED
0849- 8E 9E 09          STX D
                *--------------------------------
084C- A2 00             LDX #0          FOR E=1 TO 9
084E- E8        LOOP.E  INX
084F- 5E AC 09          LSR FLAGS,X
0852- 90 15             BCC NEXT.E      ...DIGIT ALREADY USED
0854- 8E 9F 09          STX E
0857- E0 05             CPX #5
0859- F0 08             BEQ FIX.E       E can't be 5
                *--------------------------------
085B- 20 A6 08          JSR COMPUTE.ABC
085E- B0 03             BCS FIX.E       ...NOT AN ANSWER
0860- 20 58 09          JSR PRINT
                *--------------------------------
0863- AE 9F 09  FIX.E   LDX E           NEXT E (1...9)
0866- FE AC 09          INC FLAGS,X
0869- E0 09     NEXT.E  CPX #9
086B- 90 E1             BCC LOOP.E
                *--------------------------------
086D- AE 9E 09  FIX.D   LDX D           NEXT D (1...7)
0870- FE AC 09          INC FLAGS,X
0873- E0 07     NEXT.D  CPX #7
0875- 90 CC             BCC LOOP.D
                *--------------------------------
0877- AE A3 09  FIX.I   LDX I           NEXT I (2...9)
087A- FE AC 09          INC FLAGS,X
```

```
087D- E0 09     1700 NEXT.I CPX #9
087F- 90 B3     1710        BCC LOOP.I
                1720 *--------------------------------
0881- AE A2 09  1730 FIX.H  LDX H          NEXT H (F+1...9)
0884- FE AC 09  1740        INC FLAGS,X
0887- E0 09     1750 NEXT.H CPX #9
0889- 90 9E     1760        BCC LOOP.H
                1770 *--------------------------------
088B- AE A1 09  1780 FIX.G  LDX G          NEXT G (2...9)
088E- FE AC 09  1790        INC FLAGS,X
0891- E0 09     1800 NEXT.G CPX #9
0893- B0 03     1810        BCS FIX.F
0895- 4C 19 08  1820        JMP LOOP.G
                1830 *--------------------------------
0898- AE A0 09  1840 FIX.F  LDX F          NEXT F (3...8)
089B- FE AC 09  1850        INC FLAGS,X
089E- E0 08     1860 NEXT.F CPX #8
08A0- B0 03     1870        BCS END
08A2- 4C 0C 08  1880        JMP LOOP.F
                1890 *--------------------------------
08A5- 60        1900 END    RTS            END OF PROGRAM
                1910 *--------------------------------
                1920 COMPUTE.ABC
08A6- A2 03     1930        LDX #D-DIGITS      DE = D*10+E
08A8- 20 4B 09  1940        JSR CALC.XX
08AB- 8D A4 09  1950        STA DE
08AE- A2 05     1960        LDX #F-DIGITS      FG = F*10+G
08B0- 20 4B 09  1970        JSR CALC.XX
08B3- 8D A5 09  1980        STA FG
08B6- A2 07     1990        LDX #H-DIGITS      HI = H*10+I
08B8- 20 4B 09  2000        JSR CALC.XX
08BB- 8D A6 09  2010        STA HI
                2020 *---WXYZ = FG * HI---------------
08BE- A0 08     2030        LDY #8         multiply 8-bits by 8-bits
08C0- A9 00     2040        LDA #0
08C2- 4E A5 09  2050 .1     LSR FG         get next bit of multiplier
08C5- 90 04     2060        BCC .2         bit = 0
08C7- 18        2070        CLC            bit = 1
08C8- 6D A6 09  2080        ADC HI         Add multiplicand
08CB- 6A        2090 .2     ROR            Shift product hi-byte
08CC- 6E A7 09  2100        ROR WXYZ             product lo-byte
08CF- 88        2110        DEY            Next bit
08D0- D0 F0     2120        BNE .1         ...more to go
08D2- 8D A8 09  2130        STA WXYZ+1     store hi-byte of product
08D5- 8D AA 09  2140        STA ABC+1
08D8- AD A7 09  2150        LDA WXYZ
08DB- 8D A9 09  2160        STA ABC
                2170 *---ABC = WXYZ / DE--------------
08DE- A0 10     2180        LDY #16
08E0- A9 00     2190        LDA #0
08E2- 18        2200        CLC
08E3- 2E A9 09  2210 .3     ROL ABC
08E6- 2E AA 09  2220        ROL ABC+1
08E9- 2A        2230        ROL
08EA- CD A4 09  2240        CMP DE
08ED- 90 03     2250        BCC .4
08EF- ED A4 09  2260        SBC DE
08F2- 88        2270 .4     DEY
08F3- D0 EE     2280        BNE .3
08F5- 2E A9 09  2290        ROL ABC        ...final bit into quotient
08F8- 2E AA 09  2300        ROL ABC+1
08FB- 8D AB 09  2310        STA REM
08FE- C9 01     2320        CMP #1         If any remainder, not a valid answer
0900- B0 47     2330        BCS .9
                2340 *---Check digits of ABC----------
0902- A2 FF     2350        LDX #-1
0904- AD A9 09  2360        LDA ABC
0907- AC AA 09  2370        LDY ABC+1
090A- 38        2380 .45    SEC
090B- E8        2390 .5     INX            COUNT 100'S
090C- E9 64     2400        SBC #100
090E- B0 FB     2410        BCS .5
0910- 88        2420        DEY
0911- 10 F7     2430        BPL .45
0913- 69 64     2440        ADC #100       CORRECT FOR OVER-SUBTRACTION
0915- BC AC 09  2450        LDY FLAGS,X    SEE IF DIGIT ALREADY USED
0918- F0 2F     2460        BEQ .9         ...YES, NO NEED TO LOOK FURTHER
091A- 8E 9B 09  2470        STX A
091D- A2 FF     2480        LDX #-1
091F- 38        2490        SEC
```

Insist on GSRAM™ When You Buy Your IIGS™

Expand the IIGS RAM and ROM with the GSRAM or GSRAM Plus with ROM Pak. Available now with 256K to 8 MEG!

GSRAM Plus™



GSRAM™

Remember the 16K cards for the II+ and the 64K cards for the IIe? At the time, that much memory seemed like a lot. But when the owners of these memory cards came to us for more memory, many had to throw away their smaller Apple memory cards or try to sell them. Most of our customers told us that had they known about Applied Engineering's larger memory cards when they bought their Apple, they would have purchased them at the same time.

GSRAM and GSRAM Plus are available now, allowing up to 8 MEG of memory expansion. That's 8 times the memory capacity of Apple's card and just look at the benefits that only GSRAM and GSRAM Plus have over Apple's card:

- Lower cost
- Has 6 RAM banks (Apple's card has 4)
- Has memory expansion port
- Has ROM expansion port
- No configuration blocks to set
- No soldered in RAM chips
- Expandable to 8 MEG
- Expands AppleWorks internal limits
- Built-in Hi-Res self-diagnostic software
- 5 year hassle free warranty (Apple has a 90 day warranty)
- Made in USA

GSRAM for More AppleWorks Power

Only GSRAM and GSRAM Plus eliminate AppleWorks internal memory limits, increasing the maximum number of records available from 6,000 to over 25,000 and only GSRAM and GSRAM Plus increase the number of lines permitted in the word processing mode from 6,000 to over 15,000. And only GSRAM and GSRAM Plus offer a built-in printer buffer so you can continue using Appleworks while your printer is printing. GSRAM and GSRAM Plus even expand the number of lines in the clipboard from 255 to 2047 and will auto segment large files so they can be saved on two or more disks. You can even have Pinpoint or Macroworks and your favorite spelling checker in RAM for instant response. GSRAM and GSRAM Plus will even display the time and date right on the AppleWorks screen. Nothing comes close to enhancing AppleWorks so much.

Turn Your IIGS into a Giant

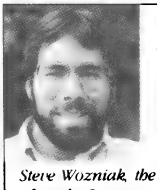
Simply plug GSRAM into the IIGS memory expansion slot and you've got up to *8 megabytes* of RAM at your fingertips—all of it instantly and automatically recognized by the IIGS. GSRAM is compatible with all IIGS software, including AppleWorks, as well as BASIC®, ProDos, DOS 3.3, PASCAL®, "C" and CP/M®.

Grow by Bytes or Megabytes

We offer GSRAM in two configurations so you can increase your memory 256K at a time (GSRAM) or a megabyte at a time (GSRAM Plus). Both offer full compatibility, lower cost than other boards, and easy expandability. And both are extremely low in power consumption. A fully expanded GSRAM operates at only 375 ma, and GSRAM Plus at only 270 ma (even with 6 megabytes on board!)

GSRAM—for Normal Memory Requirements

GSRAM is available with 256K, 512K, 1 MEG or 1.5 MEG of memory already on board. If you don't need the full 1.5 MEG now, you can choose a GSRAM with less memory and expand it up to 1.5 MEG in the future—or upgrade to GSRAM Plus for a small charge.



"In quality, performance, compatibility, expandability and support, Applied Engineering's GSRAM and GSRAM Plus are number one."

Steve Wozniak, the creator of Apple Computer

With an optional piggyback card, you can expand GSRAM even higher than 1.5 MEG! (Other cards are only expandable to 1 MEG.)

GSRAM Plus—for Growing by Leaps and Bounds

GSRAM Plus is the first Apple memory card to use 1 MEG RAM chips on the main board. It's available with 1 to 6 MEG on board. If you don't need the whole 6 MEG now, you can buy a GSRAM Plus with less memory and easily expand it in the future.

GSRAM Plus can be expanded up to 8 MEG with an optional piggyback card.

Easy Expansion

Both GSRAM and GSRAM Plus use standard RAM chips that are readily available and just plug right in. So unlike other cards, you'll find expanding your GSRAM or GSRAM Plus easy, convenient and very economical. And with our optional ROM expansion module you can even increase the IIGS's ROM space and all in just one slot.

Quality and Support of the Industry Leader

Applied Engineering is the largest supplier of Apple peripherals in the world. We *invented* the first large RAM cards for the Apple. With a 5-year "no-hassle" warranty and outstanding technical support, you can be sure GSRAM and GSRAM Plus will deliver the performance you're looking for—or return them within 15 days for a full refund.

GSRAM with 256K	$169
GSRAM with 512K	$219
GSRAM with 1 MEG	$299
GSRAM with 1.5 MEG	$379
GSRAM with 2-8 MEG	CALL
GSRAM PLUS with 1 MEG	$459
GSRAM PLUS with 2 MEG	$759
GSRAM PLUS with 3-8 MEG	CALL

Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 6¼% sales tax. Add $10.00 outside U.S.A.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

CP/M is a registered trademark of Digital Research, Inc.

```
0920- E8             .6      INX            COUNT 10'S
0921- E9 0A                  SBC #10
0923- B0 FB                  BCS .6
0925- 69 0A                  ADC #10        CORRECT FOR OVER-SUBTRACTION
0927- BC AC 09               LDY FLAGS,X    SEE IF DIGIT ALREADY USED
092A- F0 1D                  BEQ .9         ...YES, NO NEED TO LOOK FURTHER
092C- EC 9B 09               CPX A
092F- F0 18                  BEQ .9
0931- 8E 9C 09               STX B
0934- AA                     TAX
0935- BC AC 09               LDY FLAGS,X    SEE IF DIGIT ALREADY USED
0938- F0 0F                  BEQ .9         ...YES, NOT A SOLUTION
093A- EC 9B 09               CPX A
093D- F0 0A                  BEQ .9
093F- EC 9C 09               CPX B
0942- F0 05                  BEQ .9
0944- 8E 9D 09               STX C          ...NO, WE HAVE A SOLUTION
                     *--------------------------------
0947- 18                     CLC
0948- 60                     RTS
0949- 38             .9      SEC
094A- 60                     RTS
                     *--------------------------------
                     CALC.XX
094B- BD 9B 09               LDA DIGITS,X
094E- 0A                     ASL
094F- 0A                     ASL
0950- 7D 9B 09               ADC DIGITS,X
0953- 0A                     ASL
0954- 7D 9C 09               ADC DIGITS+1,X
0957- 60                     RTS
                     *--------------------------------
                     PRINT
0958- A0 00                  LDY #0
095A- B9 88 09       .1      LDA FORMAT,Y
095D- 30 06                  BMI .2
095F- AA                     TAX
0960- BD 9B 09               LDA DIGITS,X
0963- 09 B0                  ORA #"0"
0965- 20 ED FD       .2      JSR COUT
0968- C8                     INY
0969- C0 13                  CPY #FMT.SZ
096B- 90 ED                  BCC .1
                     *--------------------------------
096D- AD 00 C0               LDA $C000
0970- 10 15                  BPL .5
0972- 8D 10 C0               STA $C010
0975- C9 8D                  CMP #$8D
0977- F0 0C                  BEQ .4
0979- AD 00 C0       .3      LDA $C000
097C- 10 FB                  BPL .3
097E- 8D 10 C0               STA $C010
0981- C9 8D                  CMP #$8D
0983- D0 02                  BNE .5
0985- 68             .4      PLA            POP RETURN, SO RTS QUITS
0986- 68                     PLA
0987- 60             .5      RTS
                     *--------------------------------
0988- 00 01 02       FORMAT .HS 00.01.02       ABC
098B- A0 D8 A0              .AS -" X "         times
098E- 03 04                 .HS 03.04          DE
0990- A0 BD A0              .AS -" = "
0993- 05 06                 .HS 05.06          FG
0995- A0 D8 A0              .AS -" X "         times
0998- 07 08                 .HS 07.08          HI
099A- 8D                    .HS 8D
13-                  FMT.SZ .EQ *-FORMAT
                     *--------------------------------
                     DIGITS
099B-                A       .BS 1
099C-                B       .BS 1
099D-                C       .BS 1
099E-                D       .BS 1
099F-                E       .BS 1
09A0-                F       .BS 1
09A1-                G       .BS 1
09A2-                H       .BS 1
09A3-                I       .BS 1
                     *----------------
09A4-                DE      .BS 1
09A5-                FG      .BS 1
09A6-                HI      .BS 1
09A7-                WXYZ    .BS 2
09A9-                ABC     .BS 2
09AB-                REM     .BS 1
                     *------------------
09AC-                FLAGS   .BS 10
                     *------------------
```

Problem with IIgs 3.5"-Drive Firmware.....Bob Sander-Cederlof

The firmware in the IIgs for the 3.5"-drive includes the so-called "Protocol Converter" interface. We discussed this interface a little in previous issues, especially in the May 1986 article showing how to make DOS 3.3 work with the little drives.

Last month we reported a fix you need to make to our DOS 3.3 patches if you want it to work on a IIgs. Then last week I found out about another IIgs-related problem.

The 3.5"-drive firmware in the IIgs, when you call it through the Protocol Converter interface, stomps on four pagezero locations which are used by other programs. Locations $57, 58, 59, and 5A all are used without any concern for how they might already be in use. The firmware makes a great effort to save and restore all sorts of other locations, but these it just walks over, kicking sand like the big bully at the beach.

We first noticed the problem when using the S-C Macro Assembler with our 3.5" version of DOS 3.3. After any commands using the 3.5" drive, such as LOAD, SAVE, CATALOG, or whatever, the line number INCREMENT would be cleared to zero. That value is kept by S-C Macro in $5A, which the 3.5" firmware zeroes. This means after any disk operation you need to type INC 10, or INC with whatever increment you want to use. Luckily, the other three bytes are not actively in use during a disk operation.

In Applesoft these locations hold temporary string descriptors. I think they are only used while executing one statement, so it is possibly all right to share them with the firmware.

Apple probably is not going to do anything about this problem, because supposedly only NEW software would be coming in through the Protocol Converter interface. The ProDOS interface does not have any problem, because it either does not use those locations or it saves-restores them properly.

I think we probably are going to have to be the ones to change. Either we have to add code to our Unidisk DOS 3.3 to save and restore those four bytes, or we have to change the patch to use the ProDOS interface for reading and writing blocks rather than the Protocol Converter. The trouble with the latter approach is that the ProDOS interface requires the use of locations $42-$47, and four of these are already in use by higher levels of DOS 3.3! It is hard to win at this game. Either way we end up needing to save and restore four bytes.

Until we decide one way or the other, we need to at least come up with a fix for our Unidisk DOS 3.3 that will allow you to use it with the S-C Macro Assembler without losing yur INCrement every time you turn around. Of the four clobbered bytes, only $5A is critical to the assembler. I found a way to save enough bytes in Bill's code to slip this in. It had to be "slipped" in, so that the references to the DOS patches made in Bill's boot program do not have to be changed.

Lines 1970-2020 in Bill's Unidisk driver were:

```
1970    sta block
1980    ldy #5
1990    lda (iob.ptr),y
2000    lsr
2010    ora block
2020    sta block
```

Replace those lines with the following, which take the same space:

```
1970    asl
1980    ldy #5
1990    ora (iob.ptr),y
2000    ror
2010    sta block
2012    lda $5a
2014    pha
2016    nop
2020    nop
```

This performs the same function as the original code, but adds the feature of saving location $5A on the stack. At the exit we need to restore $5A from the stack, so look at lines 3100-3110 of Bill's program. They were:

```
3100    sta (iob.ptr),y
3110    rts
```

Change them to:

```
3100    jmp bobs.patch
```

Then insert new code as follows:

```
3202 bobs.patch
3203    sta (iob.ptr),y
3204    pla
3205    sta $5a
3206    rts
```

I entered the above patches by using the monitor peeking and poking commands, and they do seem to work correctly. Here are the commands I used:

```
00/BF8E:91 48 68 85 5A 60
00/BF82:4C 8E BF
00/BED6:0A A0 05 11 48 6A 8D 89 BF A5 5A 48 EA EA
```

You could add these commands to the EXEC file which loads the IIgs version of the S-C Macro Assembler.

In going over Bill's code while preparing this article, I also noticed that his line 1860, "STA LAST.BLOCK", should be "STA LAST.BLOCK+1". You probably should make this change regardless of what kind of computer you are using.

IIgs Tool Set Version Numbers............Bob Sander-Cederlof

According to Apple's published specifications, all IIgs tool sets are supposed to contain a function which returns a version number. Furthermore I think we can expect that the version numbers will be updated whenever any changes are made to the tools. Hopefully, programs which use the tools can first check to see if the version in ROM or RAM is up-to-date enough to be used.

It would be nice to be able to run a little program which would list the current version number for all installed tools. I wrote such a program, and it is shown below.

Since I am running under control of the S-C Macro Assembler Version 2.0, my program will receive control from the emulation mode. Therefore lines 1110-1130 get me into full native mode so that I can call tools. When I am finished, lines 1280-1290 put me back in emulation mode and return to the S-C Macro control.

The body of the program is a loop to call each of the tool sets from 1 to $20. Function number 4 of each tool is supposed to return the version number. This function requires one two-byte parameter space on the stack, in which to return the version number. Lines 1170-1190 set up the call and call a tool, and lines 1200-1220 store any error code returned as well as the version number. If the tool is not installed we will get an error code of 0002.

To display the results I wrote a little subroutine which runs in emulation mode. This is convenient, since all of our friendly monitor ROM I/O routines left over from the old days are only callable from emulation mode. There are probably some nice tools in the IIgs to take the place of COUT, PRBYTE, CROUT, and others, but these old friends are certainly easier to use and to remember.

My display routine prints an encoded string. Any bytes in the string which have the high bit on are printed as ASCII characters by calling COUT at $FDED. Any byte in value between $01 and $7F is treated as an index into page zero. The contents of the indexed location are printed out in hexadecimal by calling PRBYTE at $FDDA. A $00 byte in the string marks the end.

PRINT.FORMAT is called with the Y-register pointing just before the first character to be printed. This arrangement lets me break after printing the version number, and then just call PRINT.FORMAT again to print the error number when there is one.

Note at line 1370 I get the status byte I pushed at line 1320 by using a stack-relative LDA. Then I used LSR to shift bit 0, which was the carry bit, into C. Why didn't I use PLP followed by a PHP here? It would save one byte and have the same net effect, right?

Well I tried that, and it didn't work. The reason is that the

PLP-PHP is inside my emulation mode subroutine. In emulation mode it is impossible to keep the status bits in the m- and x-bit positions in a zero state. Regardless of what they were when I pushed them at line 1320, after a PLP-PHP operation in emulation mode they will both be 1's. This undid the calling program, which believed it was still in full 16-bit mode. Zap! Pow! Wham! Beep! So I used the safer approach.

If you want to adapt the display subroutine for other purposes and use it in an older Apple that doesn't have a 65816 in it, you can go back to the PLP-PHP method. You will also need to change the BRA opcodes at lines 1540 and 1560 to JMP's.

```
                        1000 *SAVE S.TOOL.VERSIONS
                        1010 *--------------------------------
                        1020        .OP 65816
                        1030 *--------------------------------
01-                     1040 TOOL    .EQ $01,02
03-                     1050 VERSION .EQ $03,04
05-                     1060 ERROR   .EQ $05,06
                        1070 *--------------------------------
                        1080 *   Program to display Version Numbers of All Tools
                        1090 *--------------------------------
                        1100 DVN
000800- 18              1110        CLC
000801- FB              1120        XCE
000802- C2 30           1130        REP ##$30    "Full" Native mode
000804- A9 01 04        1140        LDA ##$0401  For TOOL = $0401 to $0420
000807- 85 01           1150        STA TOOL
                        1160 *--------------------------------
000809- F4 00 00        1170 .1     PEA 0        Make room for version number
00080C- A6 01           1180        LDX TOOL     Tool # and function
00080E- 22 00 00 E1     1190        JSR $E10000  Go get it!
000812- 85 05           1200        STA ERROR    Any error code
000814- 68              1210        PLA          Tool version #
000815- 85 03           1220        STA VERSION
000817- 20 25 08        1230        JSR DISPLAY.RESULTS
00081A- E6 01           1240        INC TOOL     Next tool set
00081C- A5 01           1250        LDA TOOL     ...only go so far...
00081E- C9 21 04        1260        CMP ##$0421
000821- 90 E6           1270        BCC .1       ...not there yet
000823- FB              1280        XCE          ...that is far enough
000824- 60              1290        RTS
                        1300 *--------------------------------
                        1310 DISPLAY.RESULTS
000825- 08              1320        PHP
000826- 38              1330        SEC
000827- FB              1340        XCE
000828- A0 FF           1350        LDY #-1
00082A- 20 39 08        1360        JSR PRINT.FORMAT
00082D- A3 01           1370        LDA 1,S      GET STATUS BYTE
00082F- 4A              1380        LSR          TO SEE IF THERE WAS AN ERROR
000830- 90 03           1390        BCC .1       ...NO ERROR
000832- 20 39 08        1400        JSR PRINT.FORMAT
000835- 18              1410 .1     CLC
000836- FB              1420        XCE
000837- 28              1430        PLP
000838- 60              1440        RTS
                        1450 *--------------------------------
                        1460 PRINT.FORMAT
000839- C8              1470 .1     INY
00083A- B9 4F 08        1480        LDA FORMAT,Y
00083D- 30 0A           1490        BMI .2
00083F- F0 0D           1500        BEQ .3       ...END OF FORMAT
000841- AA              1510        TAX
000842- B5 00           1520        LDA 0,X
000844- 20 DA FD        1530        JSR $FDDA
000847- 80 F0           1540        BRA .1
000849- 20 ED FD        1550 .2     JSR $FDED
00084C- 80 EB           1560        BRA .1
00084E- 60              1570 .3     RTS
```

```
                      *--------------------------------
00084F- 8D            FORMAT .HS 8D
000850- D4 EF EF EC
000854- A0                   .AS -"Tool "
000855- 01                   .DA #TOOL
000856- A0 F6 E5 F2
00085A- F3 E9 EF EE
00085E- A0                   .AS -" version "
00085F- 04 03                .DA #VERSION+1,#VERSION
000861- 00                   .HS 00
                      *
000862- A0 E5 F2 F2
000866- EF F2 A0             .AS -" error "
000869- 06 05                .DA #ERROR+1,#ERROR
00086B- 00                   .HS 00
                      *--------------------------------
```


Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are $1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.